(12)

CHI Systems, Inc.
100 Edward Burns Place
Goleta, CA 93117
(805) 964-8868

May 1982

CHI-5 MICRO-PROGRAMMING REFERENCE MANUAL

Quarterly Technical Report

10 February 1982 - 10 May 1982

PRINCIPAL INVESTIGATER: Dr. Glen J. Culler

PROJECT SCIENTIST: Dr. Judith B. Bruckner

This research was support by the Defense Advanced Research Projects Agency under ARPA Order No. 3625 Contract No. MDA 903-82-C-0136.

CHI-5 MICRO-ASSEMBLER

CONTENTS

TABLES OF FIGURES

CHI-5 MICRO PROGRAMMING REFERENCE MANUAL

1. GENERAL INFORMATION.

1.1 INTRODUCTION

The CHI-5 is a general purpose computer whose inner structure is that of an array processor. This somewhat unique architecture allows the maintainence of good performance for standard computational processes while accomplishing high performance for array processes. As a result, the CHI-5 is an array processor which can act as its own host, or it can work, in conjunction with a host, on a wide variety of applications. The architecture of the CHI-5, with its choice of multiple memories and accessibility to the micro instructions, gives the sophisticated user the opportunity to tailor the machine for any specific problem.

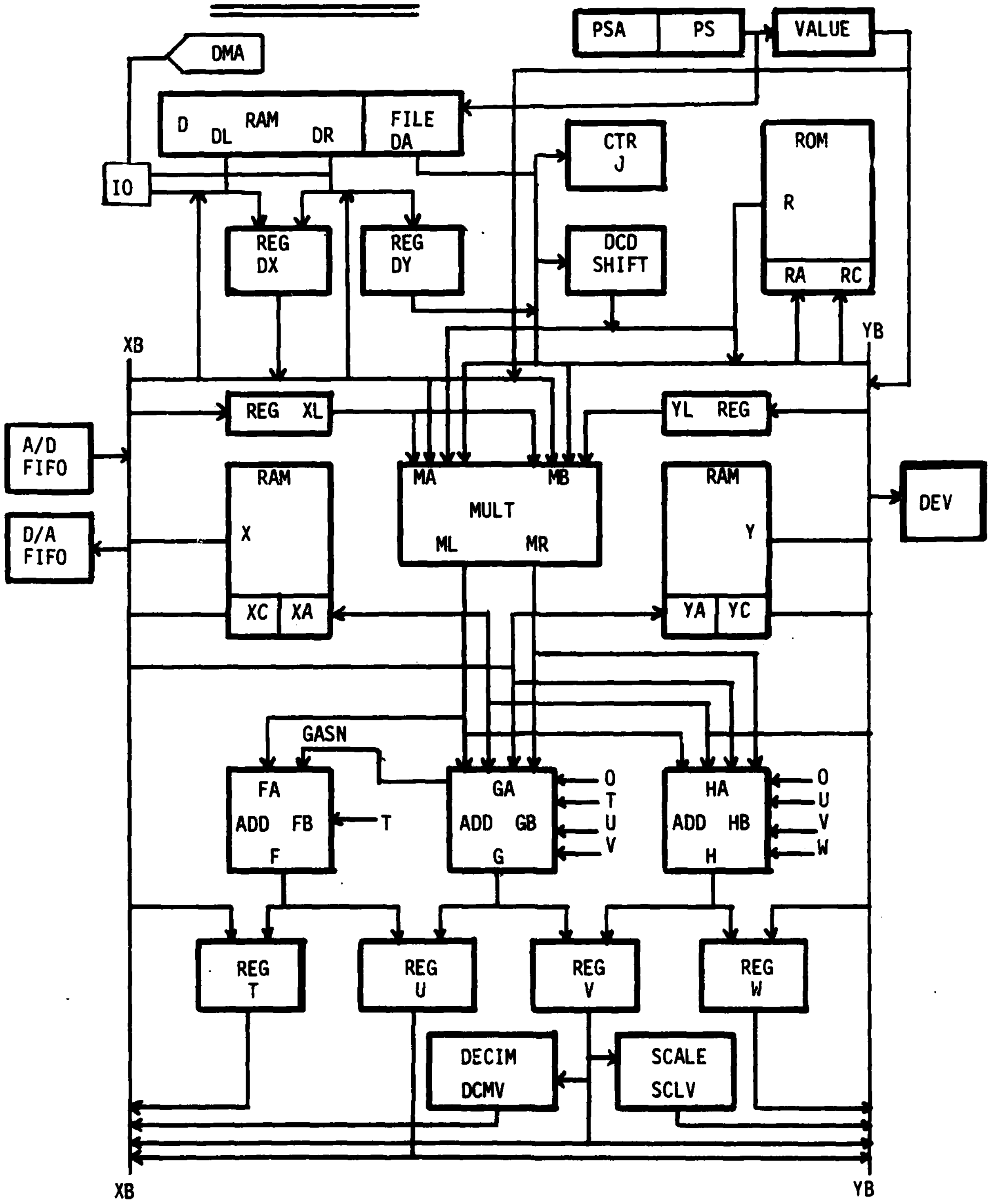
The CHI-5, as delivered, has a powerful complement of macro operations, which are the fundamental building blocks of all CHI-5 programs. This macro operation set contains the usual arithmetic operations on single numbers, as well as a variety of array processes whose arguments are lists of numbers. Each macro is implemented by a package of micro instructions in read-only-memory which in turn control the hardware.

These standard macros are sufficient for most scientific problems arising from digital signal processing, matrix arithmetic and numerical simulation. However, the user may wish to augment the standard set of operations with additional macros appropriate to his/her specific application. The ability for the user to define his own macros and to integrate them into a CHI-5 program with existing standard macros is a special feature of this machine. This manual is specifically designed to provide the tools needed to enable the user to implement this facility.

1.2. SYSTEM ARCHITECTURE.

In Figure 1 which follows, we show the architectural relationships of the family of logical modules comprising the CHI-5.

FIGURE 1. CHI-5 ARCHITECTURE



The hardware of the CHI-5 can be subdivided into 6 major modules. Brief descriptions of these modules are given below. The modules and their major submodules are described in more detail in section 2.

1. MICRO PROGRAM FACILITY

The program memory sequencer provides the controls and addressing capability for the micro program memory (PS). The sequencer performs sequential access to PS memory as well as conditional and unconditional branches. This unit also provides for microcode subroutines with a 16 deep stack file. A loop counter is also included in the sequencer for ease of repeating a microcode sequence. Interrupt signals from I/O devices cause an automatic branch to an interrupt routine.

The PS memory contains the executable microcode and is implemented as an 80-bit wide by 3K deep memory. The first and 3rd K is in ROM and the middle 1K is RAM. The loading of PS memory RAM is a special mode in which microcode previously downloaded from a host to D memory is then transferred into the program memory RAM for eventual execution. This mode not only allows for microcode checkout, but permits the CHI-5 to be reprogrammed for a variety of user applications.

2. ARITHMETIC PROCESSING UNIT (APU).

The heart of the APU consists of a multiplier, 3 16-bit adders - F, G and H, and 4 accumulator registers - T, U, V, W. The adders and accumulators can be used separately or certain of the adders can be linked to allow for 32-bit and 48-bit operations. During each system clock period, 1 multiply and up to 3 adder operations can be initiated and completed.

3. MAIN MEMORY

A major element of the CHI-5 system is the D memory which contains macro instructions, information destined for the array memories, information destined for micro-control memory and information destined for the arithmetic processing unit (APU). This memory can be accessed as 16-bit or 32-bit words.

The D memory controller provides the addressing capability for the D memory. A file of 16 D memory address registers is provided. Eight of these registers are used for I/O operations. The remaining eight registers are used for internal functions such as CHI-5 macro-instruction program counter, macro program stack pointer and data addressing.

4. ARRAY MEMORIES.

There are two random access array memories each consisting of 1024 16-bit words. They are referred to as the X and Y memories. These provide large scratch memory for the arithmetic processing unit, as well as buffers for data to be processed.

There is, in addition, a 32-bit wide, read-only, table memory, R, which holds commonly used constants and coefficients (e.g. sines and cosines).

5. BUSES

Two buses, the X-bus and Y-bus are the major communications links between logical modules. The X and Y buses inter-connect the X and Y memories, the APU, the DX and DY registers of the main memory module, the VALUE register and loop counter of the microcode program controller, the DEVICE register, serial interface, and analog FIFOs of the I/O module, and PS memory during a microcode load or store operation.

6. IO FACILITY

The major components of the IO facility are:

1) A host interface which provides for block transfer, on a direct memory access basis, between memories of the CHI-5 and its host.

2) An analog interface which provides paired A/D and D/A converters and filters operating at a fixed 8 KHZ sampling rate.

3) A dual serial interface for attachment of RS232C-compatible devices (e.g. terminal, modem, serial printer).

1.3. MICROCODE INSTRUCTION FIELDS

A micro-instruction contains a list of operations to be performed in parallel by the CHI-5 in a single system clock period (250 ns). Each operation is associated with particular hardware elements and with a set of instruction 'fields' (a field is a contiguous group of bits). Any combination of operations is allowed in a single instruction provided there are no 'field conflicts', i.e. conflicting uses of hardware elements.

FIGURE 2. CHI-5 MICRO OPERATIONS CHART

CODE	PSA	XA	X	YA	Y	G	D
0	SEQ	NOOP	NOOP	NOOP	NOOP	0	READ
1	IF STAT=0	INC XA	XB→X	INC YA	YB→Y	GB-ML	WRITE
2	IF STAT=1	INC XA(2)	XB=X	INC YA(2)	YB=Y	ML-GB	READ(#)
3	NA	DEC XA	XB→XC	DEC YA	YB→YC	ML+GB	WRITE(#)
4	IF EQ	DEC XA(2)	XB=XC	DEC YA(2)	YB=YC	|XB|+GB	READ,INC DA(YB)
5	IF LT	XC→XA	XA→XC	YC→YA	YA→YC	GB-XB	WRITE,INC DA(YB)
6	IF GT	INC XA(XC)	XA→XC,XC→X	INC YA(YC)	YA→YC,YC→Y	XB-GB	READ(#),INC DA(YB)
7	GOTO	INC XA(XC+1)	XA→XC,XB=XC	INC YA(YC+1)	YA→YC,YB=YC	XB+GB	WRITE(#),INC DA(YB)
8	EXEC MACRO	INC XA(XC-1)	XC→X	INC YA(YC-1)	YC→Y	GB+0	READ,YB→DA
9	CALL	DEC XA(XC)	X→XC	DEC YA(YC)	Y→YC	GB-MR	WRITE,YB→DA
A	NA	DEC XA(XC-1)	XA→XC,XB→X	DEC YA(YC-1)	YA→YC,YB→Y	MR-GB	READ(#),YB→DA
B	DECJ,IF J≠0	DEC XA(XC+1)	XA→XC,X→XB	DEC YA(YC+1)	YA→YC,Y→YB	MR+GB	WRITE(#),YB→DA
C	SEQ,DEC J	YB→XA	X→XC,XB=X	XB→YA	Y→YC,YB=Y	|YB|+GB	YB→DA
D	SEQ,YB→DEV	INC XA(YB)	XC→X,XB=XC	INC YA(XB)	YC→Y,YB=YC	GB-YB	YB=DA
E	SEQ,YB→J	DEC XA(YB)	XB→X,XB→XC	DEC YA(XB)	YB→Y,YB→YC	YB-GB	YB→DA(#)
F	RTN	CLR XA	XA→XC,XC→X, XB=XC	CLR YA	YA→YC,YC→Y, YB=YC	YB+GB	YB=DA(#)

CODE	XB	YB	TU	VW	F	H	M	IO
0	NOOP	NOOP	NOOP	NOOP	0	0	:22	INT HOST
1	VALUE	VALUE	F→T	H→W	T-ML	HB-HA	:P2	CLR Sbit
2	DX	DY	G→U	G→V	ML-T	HA-HB	:2P	CLR S
3	–	SCLV	F→T,G→U	H→W,G→V	T+ML	HA+HB	:PP	XB=IO
4	DCMV	R	XB→T,G→U	YB→W,G→V	T+0	HA XOR HB	:FR	XB→IO
5	U	U	XB→T	YB→W	T-GASN	HA OR HB	D	IOC
6	V	V	F→U	H→V	GASN-T	HA AND HB	B	ENABLE
7	T	W	XB→T,F→U	YB→W,H→V	T+GASN	HB+0	IO	DISABLE

FIGURE 3. CHI-5 MICRO OPERATIONS CHART (CONT)

CODE	TEST	RLD	GB	HA	HB	FG	GH	XL	YL	R	RA	HCI
0	SHFT	NOOP	0	ML	0	NOOP	NOOP	NOOP	NOOP	RL	NOOP	NOOP
1	G	YB→RA	U	XB	U	GCO→FCI	HCO→GCI	XB→XL	YB→YL	RR	INC RA	LGCO
2	H	YB→RC	V	MR	V							
3	GH	YB→RA&RC	T	YB	W							

MULTIPY OPERATIONS

$$\left\{\begin{array}{c}MA\\XB\\YB\\XL\\R\end{array}\right\} * \left\{\begin{array}{c}MB\\XB\\YB\\XL\\YL\end{array}\right\} : \left\{\begin{array}{c}22\\P2\\2P\\PP\\FR\end{array}\right\}$$

$$SHFT* \left\{\begin{array}{c}MB\\XB\\XL\\YL\end{array}\right\} : \left\{\begin{array}{c}22\\P2\\2P\\PP\\FR\end{array}\right\}$$

M-FIELD REMAP

M-CODE	NEW FIELD	OPERATIONS
5	D	SEE D-FIELD
6	B	SEE HCI & RLD FIELDS
7	IO	SEE IO-FIELD

D-MEMORY LAYOUT FOR MICRO-INSTRUCTIONS TO BE LOADED INTO PS-RAM

|← 32b →|

—	VALUE
PSA,TEST,R,RA,M,MULT	F,FG,G,GH,HA,H,XL,YL
XB,GB,TU,XA,X	YB,HB,VW,YA,Y
← 16b →	← 16b →

Figures 2 and 3 indicate the micro-operations that can be performed with the CHI-5. See section 4 for a complete description of the assembler syntax for the micro-operations. Explanations of the symbols used for module and operation names in Figures 1, 2 and 3 appear in the GLOSSARY which follows section 4.

For the most part, the fields of the micro instruction are directly associated with logical modules. The only remapping of instruction fields occurs in the MULT field. This field is used for multiplication operations, IO operations, D memory operations and some table memory operations (See figure 3).

In preparing a micro-program , section 4 should be used to express the desired operations in accordance with the syntax expected by the assembler. Figures 1, 2 and 3 can be used to insure that desired combinations of operations do not conflict.

1.4. CHI-5 APPLICATIONS

CHI-5 can be used as a stand-alone facility in numerous speech and other signal processing applications. The parallel operations embodied in the system architecture permit efficient computations in the algebra of bilinear forms.

Small scale microprocessor based computers provide user communication that is very successful for program preparation and small-scale problem solving. But, when any significant scientific computing is required, the arithmetic capability is simply not adequate. An enormous enhancement of performance will result by using the CHI-5 as an accelerator.

To achieve super-performance level, a micro-net of CHI-5 processors, with each node having general purpose control, a local operating system, and a micronet communication capability over a multipli-connected network could be implemented.

2. SYSTEM DESCRIPTION

In this section, descriptions are given for each of the major logical modules of the CHI-5. Explanations of micro operations are given for those operations which are not self explanatory. Any features which necessitate special programming considerations are noted.

2.1. MICRO PROGRAM FACILITY

The micro program facility provides the operation control of all the hardware modules of the CHI-5, and also provides means of sequencing micro-instructions and macro operations. It is comprised of four logical modules:

PS:	Micro program storage memory. 2048 by 80-bit words of ROM 1024 by 80-bit words of RAM
PSA:	Micro program address module.
J:	Loop control, down counter.
VALUE:	An immediate value field in each micro instruction.

2.1.1. PS

To provide operation control for the CHI-5's logical modules, the micro program store PS is broken into fields, with each field controlling a logical module. Except for the five bits comprising the MULT field, the micro instruction fields always refer to the same hardware. Thus, many operations may be performed in parallel, with each operation started and completed in one system clock period.

2.1.2. PSA

The micro program sequencing is accomplished by the use of the PSA module. In figure 4 below, we briefly define the PSA operations indicated in the PSA field of Figure 2.

FIGURE 4

PSA OPERATIONS

OP	DESCRIPTION
SEQ	PSA+1 -> PSA. While executing the present micro-instruction, fetch its successor. (Default operation for this field)
IF STAT=0 GOTO val	If the status of the device presently selected is zero, branch to the address in the value field. Otherwise sequence as above.
IF STAT=1 GOTO val	If the status of the device presently selected is one, branch to the address in the value field, otherwise sequence as above.
IF TEST $\{=,<,>\}$ 0 GOTO val	Arithmetic conditionals on the register selected by the TEST field. Branch when condition is met. See section 2.2.5.
GOTO val	val->PSA. Unconditional branch.
EXEC MACRO	This instruction must be immediately preceded by the D-operation READ(EA). In this case, EXEC MACRO causes the selected argument gatherer to be activated and the micro operation to be accessed.
CALL val	PSA+1 -> micro stack. val -> PSA. Increment the stack pointer. This is the micro-subroutine call.
DEC J	J-1 -> J. Sequence as for SEQ.
DEC J, IF J>0 GOTO val	IF J>0, val -> PSA. Otherwise sequence. In either case, J-1 -> J after test is made. Down counter loop control.
YB->DEV	Select the device whose device number is on the Y-bus. Sequence as for SEQ.
YB->J	Set J equal to the value on the Y-bus. Sequence as for SEQ.
RTN	Decrement the stack pointer. Micro stack -> PSA Micro subroutine return.

2.1.3. J

The J-register is a loop control, down counter. It is loaded by PSA field operation YB->J, and tested and decremented by PSA field operation: DEC J, IF J>0 GOTO PSA. It can also be decremented without the test. The test cannot be used following an instruction which loads J or decrements it without a test.

2.1.4 VALUE

Each micro instruction has a 16-bit value field. It may be used to specify micro program constants, device numbers, and PSA branch addresses.

2.2 ARITHMETIC PROCESSING UNIT (APU)

The APU contains a multiplier with 2 input registers, 2 auxilliary input registers and 2 output registers. The unit includes 3 adders and 4 accumulator registers. In addition, the APU contains logic to handle shifting, scaling, floating point normalization, bit reversal, and test logic for 2 of the adders.

2.2.1. MULTIPLIER

The 16-bit x 16-bit pipelined multiplier with input registers MA and MB produces a 32 bit product stored in 2 registers called ML and MR. ML (Multiply Product Left) contains the 16 most significant bits of the product; MR (Multiply Product Right) contains the least significant 16 bits.

A multiply is initiated in each instruction in which at least one of the multiplier input registers is loaded. The product is available in ML and MR in the second instruction following the one in which the multiply was initiated. The product remains valid until the second instruction after a new multiply is initiated.

Examples: Dot product of a vector of length n with itself. Assume for all three examples that UV has been initialized to 0 and XA points to the first component of the vector.

Example 1: Assume in addition that J contains n-1.

```
instrl   X * X                  "Initiate multiply X(i)*X(i)
instr2   INC XA                 "Update address
instr3   MLMR + UV -> UV,       "Partial Sum + X(i)*X(i)
         DEC J,                 "decrement counter
         IF J>0 GOTO instrl     "test: done?
```

The same computation can be accomplished in a 2 instruction loop as follows.

Example 2: Assume n in J, MA set to 0.

```
instrl   X * X,                 "Initiate X(i)*X(i)
         INC XA                 "Update address
instr2   MLMR + UV -> UV,       "Partial Sum + X(i-1)*X(i-1)
         DEC J,                 "Decrement counter
         If J>0 GOTO instrl     "Test: done?
```

1. The first time through the loop,
a) Instrl: X(1)*X(1) is initiated, (0*initialMB)->MLMR
b) Instr2: 0 + (0 * initial MB) = 0 -> UV

These 2 instructions have ´filled the pipeline´.
Each additional time through the loop,
a) Instrl: X(i) * X(i) is initiated, and
            X(i-1) * X(i-1) is latched into ML, MR.
b) Instr2: X(i-1) * X(i-1) is added to the partial sum.

The last time through the loop, an irrelevent multiplication is initiated. However, this product is not accumulated with the others, since J will decrement to 0 and cause control to pass out of the loop.

The same computation can be performed in a loop of a single instruction.

Example 3. Assume J contains n+1, MA initialized to 0 at least two instructions earlier.

```
instrl   X * X,                 "Initiate X(i)*X(i)
         INC XA,                "Update address
         MLMR + UV -> UV,       "X(i-2)*X(i-2) + Partial Sum
         DEC J,                 "Decrement counter
         IF J>0 GOTO instrl     "Test: Done?
```

The first time this instruction is executed, X(1)*X(1) is initiated and the 0 in MLMR is added to the 0 in UV. At the same time, 0*MB=0 is latched into MLMR. XA is updated to point to the 2nd component of the vector, and J is decremented to n+1.

The second time the instruction is executed, X(2)*X(2) is initiated, and the (2nd) 0 in MLMR is added to the 0 in UV. During the same system clock period, X(1)*X(1) is latched into MLMR. XA is updated to point to the 3rd component of the vector, and J is decremented to n.

These two initial executions of the instruction are required to fill the pipeline, and account for the n+1 loaded into J before the loop begins. In each succeeding execution of the instruction, a new multiply is initiated and a previous product is accumulated. In the last two executions of the instruction, irrelevent multiplies are initiated, but their corresponding products are never accumulated.

There are two auxilliary input registers to the multiplier called XL (X-latch) and YL (Y-latch). These are loaded from the X and Y bus respectively. XL can be input to either side of the multiplier; YL can act as input to the B-side only.

Possible inputs to MA fall into 4 categories:

1) Performed using a direct path: {MA, XL, RL, RR}

2) Performed using X-bus: {X, XC, DX, IO, T, DCMV}

3) Performed using Y-bus: {Y, YC, DY, FILE, W, SCLV}

4) Performed using a bus to be determined by the assembler: {value, U, V}

If 'MA' is the indicated A-side input, the value used as the A-side multiplier in the previous multiply operation is reused.

The possible inputs to the B-side of the multiplier fall into the same 4 categories:

1) Category 1: {MB, XL, YL}

2) Category 2: {X, XC, DX, IO, T, DCMV}

3) Category 3: {Y, YC, DY, FILE, W, SCLV, RL, RR}

4) Category 4: {value, U, V}

If the B-side input is ´MB´, the value used in the previous multiply operation is reused.

Note for category 4 multiplicands: The assembler assigns a bus in a manner to promote bus use consistent with the requirements of other operations which will be performed in parallel with the multiplication.

The mode of a multiplication is determined by a descriptor which indicates the nature of inputs and product. Figure 5 below shows the possible modes and their corresponding input and product types.

FIGURE 5

MULTIPLICATION OPERAND AND PRODUCT COMBINATIONS

MODE	OPERAND A	OPERAND B	PRODUCT
22	2´s comp. integer	2´s comp. integer	32-bit 2´s comp. integer in ML, MR
P2	pos. binary integer	2´s comp. integer	32-bit 2´s comp. integer in ML, MR
2P	2´s comp. integer	pos. binary integer	32-bit 2´s comp. integer in ML, MR
PP	pos. binary integer	pos. binary integer	32-bit pos. binary integer in ML, MR
FR	2´s comp. fraction	2´s comp. fraction	16-bit 2´s comp. fraction, convergent rounded in ML.

If no explicit mode is indicated, the assembler assumes mode=22.

Shifting is accomplished by multiplication by a power of 2.

For example, in the operation

```
SHIFT (P) * MB
```

where P is the value on the Y-bus, the actual multiplication performed is

```
2**Q * MB
```

where Q is the low order 4 bits of P.

This operation shifts MB left Q places, or right 16-Q places, depending on the interpretation of results. SHIFT can be used as A-input with any category 1, 2 or 4 B-input.

2.2.2. ADDERS

The CHI-5 provides 3 16-bit adders which can be used separately for 16-bit inputs, or in certain combinations for 32-bit or 48-bit inputs. The carries which provide the links between adders can be invoked implicitly for standard double adder operations (see section 4, 2.4 and 2.5), or explicitly in single adder operations.

When a subtraction is performed which includes a carry from another adder, the quantity (1-carry) is subtracted from the minuend along with the subtrahend. The assembler syntax for these operations requires this to be explicitly detailed, in order to alert the user to the exact nature of the operation being performed.

For example, `T - ML - 1 + GCO:F -> T`

This operation will be performed in the F-adder, using the carry-out from the parallel operation performed in the G-adder.

The F-adder is the most limited of the three adders, combining ML or the sign extention of the G-adder with T. Its output can be directed to either T or U.

Both the G and H-adders provide for 3 of 4 acuumulators plus zero on one side -- the B-side -- and either bus or either multiplier output register on the other. The operations performed in these adders are:

```
A+B, A-B, B-A and B+0
```

Each of these adders has an additional feature. The G-adder can perform sums where one of the inputs is the absolute value of the quantity, Q, on the X or Y bus. If Q is negative, the 1's complement = -Q-1 is used). The H-adder can perform the logical operations OR, XOR, AND.

The output of the G-adder can be directed to U or V; the output of the H-adder can be directed to V or W. In addition, an operation can be performed in either adder and its output stored nowhere, but the result used to control program sequencing.

When an operation requires the 'FG adder', the F and G adders are linked, the carry-out from G (GCO) automatically being combined with the operation performed in F. In a 'GH' operation, the H carry-out (HCO) is automatically combined with the inputs to G.

In addition, GCO can be explicitly included in F-adder operations, HCO can be explicitly included in G-adder operations and LGCO (Last-G-Carry-Out -- the carry created by the previous G-adder operation) can be explicitly included in H-adder operations. (Note: invocation of LGCO involves the remapping of the MULT field, and conflicts with a multiplication, a D memory operation or an IO operation).

The permissable inputs to the adders are shown below:

FA: {ML, GA-sign extension}

FB: {T}

GA and HA:

1) Direct path: {ML, MR}

2) X-bus: {X, XC, DX, IO, T, DCMV}

3) Y-BUS: {Y, YC, DY, DA, W, SCLV, RL, RR}

4) Bus determined by assembler: {value, U, V}

GB: {0, T, U, V}

HB: {0, U, V, W}

When an adder operation is listed in an instruction, the assembler will make the decision as to which operand will be assigned to be the A-input and which to be the B-input. In addition, if it is not explicitly stated, the assembler assigns the adder to be used to perform the operation. See section 4, subsection 2.7 for the default choices used.

The principles that the assembler uses in the assignment of inputs entail avoidence of bus use where possible and use of previously assigned buses where possible.

The assembler can postpone assignments of components which drive buses until the entire instruction has been encountered, in order to make optimal bus assignments. However, the assembler does not postpone assignment of operands to the A-side or B-side of an adder. It makes assignments in accordance with the above principles, making the best choice available, given the parallel operations which have already been processed. This may entail a certain amount of arbitrariness, particularly for the first adder operation to be processed in an instruction.

It may even, in some cases involving parallel adder operations, produce a diagnostic message indicating an assembly error for an instruction which would be possible if the assignments were optimally made.

This kind of ´assembler error´ cannot occur if the parallel operations cause all necessary bus assignments to be made. Thus, if the assembler gives a false diagnostic for such an instruction, the programmer should modify the instruction to include parallel operations for explicit bus assignments. Then the instruction will assemble without fault.

2.2.2. ACCUMULATOR REGISTERS

The four registers T, U, V, and W are used primarily as accumulators in array processing inner loops. Control of T and U is specified in the TU field: control of V and W is determined by the VW field (See Figure 2).

The T register can be loaded from the F-adder or the X-bus; The W register can be loaded from the H-adder or the Y-bus. U is loaded from the F or G adders, while V is loaded from the G or H adders.

The output of the F-adder (resp. H-adder) cannot be directed simultaneously to both the T and U registers (resp, the V and W registers).

For each of these registers, its value at the beginning of a system clock period can be used, and a new value stored, within that clock period.

Example: The following instruction interchanges the U and V registers.

U:H -> V, V:G -> U

2.2.3. BIT REVERSAL

The DCMV operator (DECIMATION OF V) provides a bit reversed version of the contents of the V register. This reversed data can be gated onto the X-bus.

In the following examples the numbers will be 8-bits long instead of the 16-bits used in the APU.

Example: Assume U contains 0110001. The operation

DCMV -> T

would cause 1000110 to be gated onto the X-bus and thence to T.

2.2.4. SCALING

The SCLV operator (SCALE of V) counts the number of leading zeros (exclusive of the sign bit) in the contents of the V register. This data is then gated onto the Y-bus. This operator is used, in conjunction with the SHIFT multiplication, in floating point normalization.

Example: Assume V register contents of 00011001...
Then SCLV is 2.

Note: The contents of V are assumed to be non-negative. If the contents of V are negative , SCLV does not produce a meaningful result. If V is zero, SCLV is 31 . If the G-adder operation in the second preceding instruction caused an overflow, SCLV is 47. These special case values were designed to facilitate floating point arithmetic and are recognized by the SHIFT test (see section 2.2.5).

2.2.5.ARITHMETIC CONDITIONALS

Program sequencing can be controlled by the outcome of tests on the results of arithmetic operations (See section 2.1, Figure 4). In all such operations, the branch is taken if the test condition is fulfilled.

The results of operations in the G, H or GH adders can be tested for being positive, negative, or equal to zero. Because of timing considerations, these tests must be made in the second instruction following the instruction in which the adder operation occured.

```
Example:    X + U: G                    "instruction 1
            any legal instruction       "instruction 2
            IF G>0 GO TO LOOP           "instruction 3
```

If the result of the operation in instruction 1 is positive, control transfers to the instruction labeled LOOP.

There are two arithmetic conditional tests which were designed to facilitate micro-programming of floating point arithmetic. These tests divide the spectrum of Y-bus values into three sets. The tests are:

1) IF SHIFT SMALL GOTO psa
2) IF SHIFT OVFL GOTO psa

Both of these operations test the value on the Y-bus at a particular time. The first tests the Y bus value of the previous instruction; the second tests the value on the Y bus during the instruction executed just prior to the previous instruction. Thus both tests can be performed on the same Y-bus quantity.

SHIFT SMALL is true if the Y-bus value in the preceding instruction is between -16 and +15. SHIFT OVFL is true if the Y-bus value in the second preceding instruction is less than -32, or bit 5 is 1. Note, 47 satisfies SHIFT OVFL and 31 satisfies neither test; both would specify a right shift of one place, or a left shift of 15 places, if used in a shift multiply. See sections 2.2.1 and 2.2.4.

2.3. MAIN MEMORY

The main memory module is used for many different purposes in the CHI-5. It contains the macro program which guides the CHI-5's operation; it can contain microcode destined to be loaded into PS memory; it can hold data destined for the APU; it can buffer data to be transferred on a block DMA basis between the CHI-5 and an external host. This memory, which is called D-memory (DATA memory), is 32 bits wide, but is accessible as either 16-bit words or as 32-bit double words. D-memory cannot be read and written in the same instruction.

CHI-5 is available with 1 - 3 units of D-memory. A unit contains 8K, 32-bit, double-words of RAM, and 1 or 2k, 32-bit, double-words of ROM. A fully expaded system has 24K double-words of RAM and 6K double-words of ROM.

The D-memory controller is organized around DA, a set of 16, 16-bit registers, whose output is an address for D-memory access. The first eight of these are reserved for data transfers between a host computer and buffers in D-memory. The second eight are for the CHI-5's exclusive use. Two of these registers serve as stack pointer and instruction counter for macro program operation. The remaining 6 are available for the use of micro-programs.

The DA index to be used in an operation is specified by enclosing it in parentheses in a READ, WRITE or DA operation. When no such quantity is present in an operation, the current (last specified) value is used. DA can be loaded from the Y-bus or can be used to drive the Y-bus.

When a new DA register is selected, the transfer mode is automatically set for single word or double word transfers. Double word mode can be selected explicitly, by appending :D to the intended DA index -- e.g. READ (9:D). It can be specified implicitly by adding 16 to the intended DA index -- e.g. READ (25). If double word transfer is not explicitly selected, and the indicated DA value is less than 16, single word transfer is assumed.

After a READ or WRITE of D-memory, the address in the currently selected register is updated. The address can be loaded from, or incremented by whatever quantity is on the Y-bus. If no explicit update operation is included in the operation, the transfer address is incremented according to the transfer mode: by 1 for single word mode; by 2 for double word mode. Note: In double word mode, the DA address must always be even.

Two 16-bit registers - DX and DY - are associated with D-memory. For single word READs, the addressed word is read into DX. For double word READs, the addressed 32-bit word is READ into DX and DY (the even address -- high order word goes into DX; the odd address -- low order word goes into DY).
Data read from D-memory is available in DX and DY in the instruction following the READ; it remains valid until one instruction after the next READ - i.e. an operation involving DX or DY which occurs in the same instruction as a READ, refers to the data from the previous READ operation.

Data transferred to the D memory uses the X-bus for single word WRITEs and both buses for double word WRITEs. Note that single word WRITEs must be utilized when the Y-bus is being used to load or increment a DA register. WRITEs do not affect DX or DY.

2.4. ARRAY MEMORIES.

Two random access array memories -- X and Y -- provide large scratch memory for the arithmetic processing unit, as well as buffers for data to be processed. These 1024 word memories are connected, via the X and Y buses respectively, to the APU. X can be read or written (but not both) in every system clock period. The same holds true for Y.

The array memory address controllers -- XA and YA -- are identical. Associated with each is an auxilliary register -- XC (resp. YC) -- to help provide the flexibility necessary for computing along rows or columns of 2-dimensional arrays. Since the workings of the two controllers are identical, only operations involving XA will be described.

In any operation, XA can be incremented, decremented, cleared (set to zero) or loaded. The increment (resp. decrement) can be 1 (default), 2, the contents of the XC register, XC+1, XC-1, or any quantity which can be gated onto the Y-bus. XA can be loaded, by a direct path, from the XC register, or it can be loaded by the value on the Y-bus.

A 32-bit wide, read-only memory, R, is provided to hold commonly used constants and coefficients (e.g. roots of unity). The R-memory is 1K long. The address, RA, from which table data is to be read can be set from the Y-bus. Similarly, the address increment, RC, is set using the Y-bus. RA and RC are registers in the R-memory address controller. The operation INC RA causes the value in RA to be incremented by the quantity in RC.

Either the left or right half of the currently addressed word of R can be gated onto the Y-bus or transferred directly to the A-side of the multiplier. The left 16 bits are referred to as RL (R-left), the right 16 bits as RR (R-right).

For each of these array memories, the address can be changed in one instruction, and data accessed at the new address in the next.

Example: The single instruction loop below sums the components of a vector. Assume the vector is in X-memory starting at 0, and n, the number of components of the vector, is in W.

```
instrl:  0 -> U,               "Initialize U (using G-adder)
         CLR XA,               "Initialize XA
         W -> J                "Initialize J counter
instr2:  X + U -> U,           "Partial sum + X(i)
         INC XA,               "i+1 -> XA
         DEC J,                "Decrementcounter
         IF J>0 GOTO instr2    "Test: done?
```

The first time instruction 2 is executed, X(0) is used. The last time, X(n-1) is used. In the instruction following instruction 2, XA=n would be used.

2.5. BUSES

The X and Y buses are the major communications links in the CHI-5. They connect the array memories to the DX and DY registers of the D-memory module and to the APU. They also connect the VALUE register and loop counter of the program controller, and the DEVICE register and analog FIFOs of the I/O module. Each bus can carry only one value at a time; however, any component connected to a bus can access it, in any instruction.

The component used to drive one of the buses can be specified explicitly -- e.g. XB = X. Alternatively, the specification can be given implicitly by listing an operation which can only be implemented by a particular choice for a bus . For example, the operation -- T * MB -- requires that T drive the X-bus.

Some operations require use of a bus to implement the operation, but either bus could do the job, for example, the operation -- U * MB. In such cases, the assembler chooses the bus which works optimally from the point of view of the operation as a whole.

For example, in the instruction

```
U * MB, T -> X
```

the assembler would assign U to drive the Y-bus, since X-bus is required to implement T -> X. But in the instruction

```
U * MB, W -> Y,
```

U would be assigned to X-bus.

Components which can access X-bus:
X, XC, T, U, V, DCMV, DX, IO, VALUE

Components which can access Y-bus:
Y, YC, U, V, W, SCLV, RL, RR, DY, DA, VALUE

2.6 I/O FACILITY

The I/O facility controls the flow of data between the CHI-5 and a host, or between the CHI-5 and external devices. This facility has five major components: Device Register, S-Register, Host Interface, Serial Line Interface, and Analog Interface. These components are described in sections 2.1 through 2.6.

The IO field of the micro-instruction is used to specify individual I/O operations. Figure 6, below, gives precise descriptions of the IO field operations shown in figure 2.

FIGURE 6

IO FIELD OPERATIONS

OP	DESCRIPTION
INT HOST	Set status bits in Command and Status Register (CSR) of Host Interface equal to the value in device register (port to be used). Set the attention bit in the CSR to interrupt the host.
CLR SBIT	Set the S-bit corresponding to the currently selected device to 0.
CLR S	Set S register = 0.
XB=IO	Gate output of currently selected device onto the X-bus. Interpretation of data depends on particular device. (Meaningful only if DEV>7).
XB->IO	Load the selected IO device with the value on the X-bus. Interpretation depends on device. (Meaningful only if DEV>7).
IOC	Device dependent operation for devices where DEV>7. Not currently implemented.
ENABLE	Enable interrupts.
DISABLE	Disable interrupts.

2.6.1 DEVICE REGISTER

The Device Register -- DEV -- selects one of 16 devices. The IO operations CLR SBIT, XB = IO, XB -> IO and IOC, and the STAT tests apply to the currently selected device. Figure 7, below, shows device assignments.

FIGURE 7

DEVICE ASSIGNMENTS

DEVICE NUMBER	USE
0	DATA PORT
1	DATA PORT
2	DATA PORT
3	DATA PORT
4	DATA PORT
5	DATA PORT
6	DATA PORT
7	COMMAND PORT
8	ANALOG INTERFACE
9	SERIAL INTERFACE
10	HOST ATTENTION
11	UNASSIGNED
12	UNASSIGNED
13	UNASSIGNED
14	UNASSIGNED
15	UNASSIGNED

Each device has a status -- STAT -- which can be tested. The tests, described in Figure 4, cause a branch conditional upon the status of the currently selected device being 0 or 1. For devices 0 - 9, STAT equals the corresponding S-bit (see section 2.6.2). For device 10, the operations test the state of the last INT HOST command. STAT is 1 if that interrupt has not been acknowledged by the host; otherwise it is 0.

The device register can be loaded using the operation -- B -> DEV -- where B is any component which can go on the Y-bus, e.g. 8 -> DEV, W -> DEV.

The device (and S-bit) selection is valid in the instruction following the one which loads the device register, for the operations:

XB = IO, XB -> IO, IOC, and CLR SBIT.

However, before the status of a newly selected device can be tested, at least one intervening instruction must be executed between the one which loads the device register and the one containing the test.

Example:

```
        instr1          8 -> DEV
        instr2          IF STAT = 1  GOTO LOC1
        instr3          IF STAT = 1  GOTO LOC2
```

Instruction 2 tests the status of the previously selected device. In the event that status was 0, instruction 3 tests the status of the analog interface (device 8).

2.6.2. S-REGISTER

The S-Register has 10 bits which correspond to ports and external devices. Loading the device register automatically causes the corresponding S-bit to be selected, for devices for devices 0 - 7.

An S-bit reflects the status of the corresponding device. If that bit is 1, it indicates that the corresponding device needs to be serviced. The nature of the service depends upon the device and the circumstances. When at least one S-bit is set, and interrupts have been enabled, an interrupt is generated during the next EXEC MACRO operation. This interrupt forces a branch to location 64.

The S-bits are set independently by the device controllers, under conditions appropriate to each device. There are two operations which can clear bits in the S-Register. The CLR SBIT operation clears a single bit, the one associated with the currently selected device. CLR S sets the entire S-Register to zero.

The S-bits can be tested with the operations

```
        IF STAT = 0  (respectively, 1)  GOTO ---
```

For devices 0 - 9, STAT is the value of the corresponding S-bit. For timing considerations involved in the STAT test, see section 2.6.1.

2.6.3. HOST INTERFACE

The primary communication path between a host computer and the CHI-5 is via DMA transfer of 16-bit words. Up to eight such transfers can be set up, each utilizing a separate port of the CHI-5. Every port has a device number (0 - 7) and an associated DA address register (see section 2.3).

All DMA transfers are initiated by the host, which provides the host memory address, word count for the transfer and a bit which determines direction of transfer. The CHI-5 cooperates by loading the D-memory address of the buffer to be used in the DA register associated with the desired port. It also loads the device register with the device number of that port.

When it has set up a specific port, the CHI-5 signals the host, via the INT HOST command, that transfer can begin. The affected DA register increments after each word is transferred. Completion of DMA transfer will cause an interrupt to the host. In addition, the S-bit corresponding to the port which was used will be set to 1.

2.6.4. SERIAL LINE INTERFACE UNIT

The CHI-5 contains two serial line interfaces. These provide an RS-232C connection to devices such as terminals, serial printers and modems, at rates up to 19.2 Kbps. Each interface can operate at its own rate, which is set by the CHI-5 program. The interface unit can interrupt the CHI-5 when either line has received a character or is ready to accept a character for transmission. The device number for this unit is 9.

An important component of the serial line interface unit is the ´Control and Output Data´ Register (CODR). The operation B -> IO, with B a component which connects to the X-bus, and DEV = 9, loads the CODR with the data to be output. It can also be used to send commands to the unit to select the line to be used, set the baud rate for the transfer, etc.

The operation XB = IO, with DEV = 9, enables the externally received data onto the low-order 8 bits of the X-bus, and unit status onto the high-order 8 bits of the X-bus. Complete specifications are described in a separate publication, "CHI-5 Serial Interface UNIT".

2.6.5. ANALOG INTERACE UNIT

The analog interface unit provides for analog-to-digital (A/D) and digital-to-analog (D/A) transfer at a fixed rate of 8 KHZ. A pair of 64-word, first-in-first-out memories (FIFOs) is part of the unit -- one for each direction.

Every 125 micro-seconds one sample is output through D/A and one sample is input through A/D. In both directions, an 8 bit, mu law, PCM code is used as the digital value. The device number for the unit is 8.

The analog interface unit provides an interrupt when the A/D FIFO is half full.

To initialize the unit, set the device register to 8. Then empty the A/D FIFO by performing XB=IO 64 times. It is not necessary to invoke D/A whenever one is performing A/D. However, if the D/A is being utilized, A/D must also be performed.

3. MICROCODE ASSEMBLER

CHI5ASM is a FORTRAN-77 program which generates relocatable, microcode modules for the CHI-5. Separately assembled modules may be linked together by the program CHI5LINK to form a single load module with fixed instruction addresses. These load modules can be incorporated into a macro program, loaded into D-memory, and subsequently loaded into PS-RAM.

Input to CHI5ASM is via a file which must be assigned to logical unit 4. Syntax for this source is described in section 3.1.

The relocatable module produced by CHI5ASM is stored in the file associated with logical unit 7. In addition, the user may select from several optional output listings. CHI-5 assembler output is discussed in section 3.2.

Section 4 contains a description of the assembler syntax for CHI-5 micro-operations.

3.1. SOURCE FILE

The text file which contains the source for the assembly is composed of instruction statements and pseudo-operations. Each instruction statement is translated by the CHI-5 assembler into a single machine instruction. The pseudo-operations produce no code, but provide information to the assembler, which guides the assembly process.

Blank lines are ignored, and comments, preceded by quotation marks ("), terminate processing of a line of text. Comments may appear in the source document in any of the following contexts:

1) Between individual instructions and/or pseudo-operations.

2) At the end of a line of text which otherwise contains part of an instruction or pseudo-operation.

3) Between lines within a single instruction or pseudo-operation.

THE TEXT DOCUMENT IS ASSUMED TO BE AVAILABLE ON LOGICAL UNIT 4 AND IS ASSUMED TO BE AT MOST 2000 LINES IN LENGTH.

3.1.1. SYNTAX

The elements of the source language are labels, keywords, constants, pseudo-operators and the 10 special symbols

´,´ ´:´ ´"´ ´+´ ´-´ ´*´ ´->´ ´(´ ´)´ ´.´

3.1.1.1. LABELS

A label is an alpha-numeric string of 1 - 16 characters, the first character of which must be alphabetic. The characters may be upper or lower case letters or the numerals 0 through 9. Labels, depending on their role in the text, can have one or more of the attributes relocatable, absolute, external, entry. In addition, a label has an associated value.

3.1.1.2. KEYWORDS, OPERATORS AND OPERANDS

CHI-5 instructions are composed of individual operations. An operation is described in terms of an operator, zero or more operands, and possibly a destination. Operands, operators and destinations correspond to elements of the CHI-5 hardware. Keywords are used to name operands, destinations and some operators. Other operators are indicated by special symbols such as ´+´ or ´->´ . Keywords, like labels, are alpha-numeric strings of characters.

3.1.1.3. NUMERIC CONSTANTS

A numeric constant is an integer in the range [-32768, 32767]. A numeric constant has the attribute absolute. Constants can be expressed in base 2, 8, 10 or 16. Default is base 10. Syntax for a constant with an explicit base designation is

´........´ BASE

where BASE is B, O, D or X respectively. (e.g.,

´1111´B = ´17´O = ´15´D = ´F´X)

3.1.1.4. LOCATION COUNTER

The location counter gives the relative address of the current instruction from the beginning of the program. It is initialized, for each program, at 0, and incremented for each instruction. It can be set to a higher value than would otherwise occur by the ´LOC´ pseudo-operation (leaving a ´gap´). However, it cannot be set back to fill a gap.

3.1.1.5. EXPRESSIONS

Expressions are used as destinations in conditional and unconditional branch operations, in DA specifications, and in EQU and LOC pseudo-operations. An expression is composed of a single term or an arithmetic combination of terms, using the operations +, -, *. Each term must be a numeric constant, a label, the location counter (denoted by ´.´) or a subexpression enclosed in parentheses. Each expression can be transformed into a fully expanded expression containing no parentheses

3.1.1.5.1. RULES FOR CODING EXPRESSIONS

1) An expression cannot contain 2 terms or 2 operators in succession.

2) An expression cannot contain more than 10 terms or 8 operators (each pair of parentheses counts as an operator).

3) An expression containing an external label must consist of a single term.

3.1.1.5.2. EVALUATION OF EXPRESSIONS

Expressions are evaluated as follows:

1) Arithmetic operations are performed left to right except that multiplication is done before addition or subtraction.

2) Parenthesized multi-term subexpressions are processed before the rest of the terms in the expression.

3.1.1.5.3. ATTRIBUTES OF EXPRESSIONS

The attributes of an expression are determined by the attributes and signs associated with the terms of the equivalent, fully expanded expression. Every term is either relocatable or absolute. In addition, a relocatable term can be external. Labels in branch statements, and the location counter, are relocatable; numeric constants are absolute.

To determine the relocatability of an expression, substitute 1 for each relocatable term and 0 for each absolute term. Evaluate the resulting expression. If it is 1, the original expression is relocatable; if it is 0, the original expression is absolute. Any other value means the expression is illegal.

An expression is external if it consists of a single, external label.

3.1.2. INSTRUCTION STATEMENTS

An instruction statement is a list of operations to be performed in parallel by the CHI-5 in a system clock period. One instruction is executed every 250 nanoseconds.

Each operation is associated with particular hardware elements and with a set of instruction 'fields' (a field is a contiguous group of bits). Any combination of operations is allowed in a single instruction provided there are no 'field conflicts', i.e. conflicting uses of hardware elements. See Figures 2 and 3 for a description of instruction fields.

The format for an instruction statement is:

```
Label[:]  operation 1, . . . ,          "comments
          . . . . . . .
          . . . . . . .
          . . . . . . .
          . . . , operation n           "comments
```

The label is optional. Each instruction is allowed only one label, even if the instruction itself occupies several lines.

The body of the statement, the operations list, is 'free-form'. Spaces and tabs can be inserted as desired for legibility. Each operation except the last MUST be followed by a comma (,). The operations list can occupy one or more lines, but a single operation must be contained in a single line.

Example:

```
Start:  X * Y:FR,               "Multiply current X,Y
        MPLMPR + UV -> UV,      "Add in previous product
        DEC XA, INC YA(2)       "update addresses
```

Specifications for CHI-5 micro-operations are presented in section 4.

3.1.3. PSEUDO-OPERATIONS

Pseudo-operations represent directions to the assembler itself. They do not generate microcode. Each pseudo operation must begin on a new line, although some pseudo-operations can occupy several lines (e.g. EXT, ENTRY). As with instruction statements, pseudo-operations may have comments. All pseudo-operations must be preceded by space[s] and/or tab[s].

Some pseudo-operations can be grouped together and represent alternative choices. Each of these groups is associated with a flag, and each flag has a default value. Such a pseudo-operation remains in force during a sequence of assemblies from a single file until another pseudo-operation from the same group is encountered. If two pseudo-operations from the same group are used within a single program, the last such pseudo-operation encountered determines the value of the associated flag for the entire program.

The pseudo-operations with associated flags, if any, in parentheses are described below. Pseudo-operations which are automatically present unless overridden (i.e. default choices) are indicated by *.

3.1.3.1. TITLE

This pseudo-operation is used to name a program and is optional. If present, it must be the first line, exclusive of blank lines and comments, of the program. The format is:

```
        TITLE name     "comments
```

The name may contain up to 16 alpha-numeric characters.

3.1.3.2. ENTRY

This pseudo-operation declares locally defined labels to be global, so that they may be referenced in separately assembled programs. The format is:

```
ENTRY   label 1, . . . ,        "Comments
. . . . . . . .
. . . . . . . . , label n       "Comments
```

3.1.3.3. EXT

This pseudo-operation declares a label to be an externally defined, global symbol. The format is

```
EXT  label 1, . . . ,      "Comments
. . . . . .
. . . . . . , label n      "Comments
```

If an external is used in an expression, it must be the entire expression.

3.1.3.4. EQU

This pseudo-operation sets the value associated with a label equal to the value of an expression. The format is

```
EQU  symbol 1 = expression 1, ... ,    "Comments
. . . . . . . . . . .
. . . . . , symbol n = expression n    "Comments
```

3.1.3.5. LOC

This pseudo-operation sets the location counter equal to the value of an expression (See section 3.1.1.4). The value of that expression must be greater than or equal to the value in the location counter at the time the ´LOC´ is encountered. The format for this pseudo-operation is:

```
LOC    expression        "Comments
```

3.1.3.6. END

This pseudo-operation signals the end of source text to be assembled (in the current program). This pseudo-operation is required, and its absence will cause an error condition.

3.1.3.7. LIST (LISTFLAG = 1) *

This pseudo-operation causes a listing of the source text to be written to logical unit 8.

3.1.3.8. NOLIST (LISTFLAG = 0)

This pseudo operation causes the listing described in section 3.1.3.7 or 3.1.3.9 to be suppressed.

3.1.3.9. INTRLV (LISTFLAG = 2)

This pseudo-operation causes the source text and hexadecimal microcode to be listed in interleaved fashion on logical unit 8. For each instruction, first the text for that instruction is displayed, followed by the microcode.

3.1.3.10. LISTOBJ (LISTFLAG = 3)

This pseudo-operation causes a listing of the object module to be written to logical unit 8. (See section 3.1.1.)

3.1.3.11. SYMBOL (SYMBOLFLAG = 1)

This pseudo-operation will cause the symbol table to be written to logical unit 8.

3.1.3.12. NOSYMBOL (SYMBOLFLAG = 0) *

This pseudo-operation suppresses listing of the symbol table.

3.1.3.13. OBJECT (OBJFLAG = 1) *

This pseudo-operation causes the hexadecimal object module to be written to logical unit 7 (IF NO ERRORS WERE DISCOVERED IN THE ASSEMBLY).

3.1.3.14. NOOBJECT (OBJFLAG = 0)

This pseudo-operation suppresses storing of the object module.

3.1.3.15. PAGE

This pseudo-operation causes a new page to be started in the source or interleaved listing (See 3.1.3.7 and 3.1.3.9). The word ´PAGE´ will not be shown in the listing.

3.1.3.16. COMPILE (COMPFLAG = 1) *

This pseudo-operation causes compilation of programs in a source file to be resumed. This pseudo-operation, and its companion NOCOMPILE, are used to assemble some programs in a file, while skipping over others.

3.1.3.17. NOCOMPILE (COMPFLAG = 0)

This pseudo-operation suppresses compilation until a ´COMPILE´ pseudo-operation is encoutered in the text. All statements between a NOCOMPILE and the next COMPILE are ignored.

3.1.3.18. EXPAND (EXPFLAG = 1)

This pseudo-operation causes an expanded listing to be written to logical unit 8 (in addition to whatever listing may be generated if LISTFLAG > 0). In this listing, the instruction fields used are specified. For each field used in the assembly of the instruction, the field name, a mnemonic for the field operation, and the code to be placed in the field are displayed. (See Instruction Statements, section 3.1.2)

3.1.3.19. NOEXPAND (EXPFLAG = 0) *

This pseudo-operation causes suppression of the expanded listing.

3.2. OUTPUT

3.2.1. OBJECT MODULE

If no errors were discovered during assembly, the object module is written in Z8 format to logical unit 7. (See sections 3.1.3.13 and 3.1.3.14). The object module consists of a number of blocks. Each has a header whose 1st (and possibly only) word is an integer code for the block type. The blocks are described in the following sections in the order in which they would appear if all were present.

3.2.1.1. TITLE BLOCK

The title block is present only if a title pseudo-operation was used. The format is:

word 1:	block type = 1
words 2 - 5:	name, 4 characters / word

3.2.1.2. ENTRY BLOCK

Each label which has been declared in an ENTRY pseudo-operation defines an entry block. The format is:

word 1:	block type = 2
words 2 - 5:	name, 4 characters / word
word 6:	instruction address

3.2.1.3. CODE BLOCK

Each LOC pseudo-operation which is followed by instruction statements defines a code block. If there are instruction statements before the first LOC, they also determine a code block. If there are several LOC statements before a block of instructions (and no instructions between the LOC statements) all the LOC statements are ignored except for the last.

Format for a code block header:

word 1:	block type = 3
word 2:	number of instructions in the block, n
word 3:	block base address

Format for the body of the block:

words 1 to 3	1st instruction
words (3n-2) to 3n	n-th instruction

The code is written 3, 32-bit words / instruction, right justified. In the list of the object module (see sections 3.1.3.10 and 7.1.6.), code is shown in hexadecimal , 1 instruction/line.

3.2.1.4. RELOCATABLE BLOCK

This block lists the addresses of instructions which reference relocatable addresses (e.g. branch instructions). When a load module is subsequently built by the linker, the instructions in the list will be adjusted to reference the appropriate fixed locations. If no references are made to relocatable addresses, the block will not be present.

Format for the block header is:

word 1:	block type = 4
word 2:	number of locations in list, n

Format for the list:

words 1 - n:	instruction address+1 reference.

3.2.1.5. EXTERNAL BLOCK

This block lists symbols which have been declared in an EXTERNAL pseudo-operation. Each symbol appears with a list of addresses of instructions which reference that symbol. At link time, these instructions will be adjusted to reference the correct fixed locations. If there are no EXTERNAL pseudo-operations or no symbol declared external is referenced in the instructions, the external block will not be present.

Format for external block header:

word 1:	block type = 5
word 2:	length of external block excluding header

Format of external block body

words 1-4	name of 1st external
word 5,....	instr address+1, for each reference -1 at end.

Similar lists are presented for each external which is referenced in an instruction. (if a symbol is declared external but is NOT referenced in the instructions, a warning message is generated. The name will NOT appear in the externals block.)

3.2.1.6. END BLOCK

This block is always present.

word 1	block type = 6

3.2.2. LISTINGS

Listings are written to logical unit 8. Various listings can be generated, depending on the pseudo-operations in force:

LIST, NOLIST, INTRLV, LISTOBJ
EXPAND, NOEXPAND

The listings produced when the corresponding pseudo-operation is in force are described in sections 3.1.3.7 to 3.1.3.9 and 3.1.3.18. If both expanded and source or interleaved listings are requested, the expanded listing will precede the other listing.

3.2.3. ERROR MESSAGES

Error and warning messages are written to logical unit 8. They follow any listings which may be present unless the pseudo-operation EXPAND is in force. In that case, error messages are inserted in the expanded listing in the section corresponding to the faulty instruction.

4. SPECIFICATION OF CHI-5 MICRO OPERATIONS

SPECIFICATION OF CHI-5 MICRO-OPERATIONS

This section presents the specifications for the micro-operations for the CHI-5. The language chosen is as high level as is compatible with full expression of the capabilities of the machine. Default cases, expressed as special provisions, have been provided to allow the most common operations to be expresed with the minimum number of qualifiers. The mnemonics in the operation definitions are described in the glossary which follows section 12.

1. MULTIPLIER OPERATIONS

A * B: C

SHIFT(D) * E: C

where A = {MA, XL, RL, RR, value, X, XC, DX, Y, YC, DY, IO, DA, T, U, V, W, DCMV, SCLV}

B = {MB, XL, YL, X, XC, DX, IO, Y, YC, DY, DA, value, T, U, V, W, DCMV, SCLV, RL, RR}

C = {22, P2, 2P, PP, FRAC}

D = {Y, YC, DY, DA, value, SCLV, U, V, W}

E = {MB, XL, YL, X, XC, DX, IO, value, T, U, V, DCMV}

1. MA * MB not allowed.
2. SHIFT(D) = 2 ** (low-order 4 bits of D)
   This operation has the effect of shifting E left D places, or right 16-D places, depending on interpretation of results.

--- --- --- --- --- ---

NOTES:
<-> indicates "is mutually replaceable by"

{ } indicates that a choice of possibilities must be made

[ ] indicates that a choice of possibilities may be made, or that the enclosed expression may be left blank.

'value' is a signed integer in the range [-32768, 32767] or an unsigned integer in the range [0, 65535].

Special provisions:

A * B <-> A * B: 22

2. ADDER OPERATIONS

The adder operation P op Q may be implementable in more than one way. For example,

U + V -> U

can be performed in either the G or H adder. In addition, U can be assigned to the A side of the adder (through either the X or Y bus) and V assigned to the B side, or V can be assigned the A side (through a bus) and U assigned to the B side (See figure 1).

In adder operations in which the adder to be used is not specified explicitly, the default cases are listed in sections 2.6 and 2.7.

The main principle used by the assembler in assigning operands to the A or B side of an adder is to minimize bus use:

1) Avoid using buses if possible.

2) If a bus must be used, try to implement the operation using a bus which has already been assigned.

2.1 F-ADDER OPERATIONS

$$\begin{matrix} \text{FA} \pm \text{T} \\ \text{T} \pm \text{FA} \end{matrix} \quad :\text{F} \rightarrow \begin{Bmatrix} \text{T} \\ \text{U} \end{Bmatrix}$$

where FA = {MPL, GASN}

Special provisions:

$$\begin{Bmatrix} \text{FA} + \text{T} \\ \text{T} + \text{FA} \end{Bmatrix} + \text{ GCO:F} \rightarrow \begin{Bmatrix} \text{T} \\ \text{U} \end{Bmatrix}$$

$$\begin{Bmatrix} \text{FA} - \text{T} \\ \text{T} - \text{FA} \end{Bmatrix} -1 + \text{ GCO:F} \rightarrow \begin{Bmatrix} \text{T} \\ \text{U} \end{Bmatrix}$$

$$\left\{\begin{matrix} 0 \\ T \end{matrix}\right\} : F \text{ -> } \left\{\begin{matrix} T \\ U \end{matrix}\right\}$$

$$T + GCO \quad :F \text{ -> } \left\{\begin{matrix} T \\ U \end{matrix}\right\}$$

2.2. G-ADDER OPERATIONS

1) $$\left\{\begin{matrix} GA \pm GB \\ GB \pm GA \end{matrix}\right\} :G \quad \left[\text{ -> } \left\{\begin{matrix} U \\ V \end{matrix}\right\}\right]$$

where GA = { MPL, MPR, value, X, XC, DX, IO, XB, Y, YC, DY, DA, T, U, V, W, DCMV, YB, SCLV, RL, RR}

AND GB = {0, T, U, V}

2) $$\left\{\begin{matrix} ABS(C) + GB \\ GB + ABS(C) \end{matrix}\right\} \left[\text{ -> } \left\{\begin{matrix} U \\ V \end{matrix}\right\}\right]$$

where C = { X, XC, DX, IO, Y, YC, DY, XB, YB, T, U, V, W, DCMV, SCLV, RL, RR}

Note: ABS(C) = 1´s complement of C if C<0; 0 otherwise.

Special provisions:

$$\left\{\begin{matrix} GA \\ 0 \\ GB \end{matrix}\right\} :G \quad \left[\text{ -> } \left\{\begin{matrix} U \\ V \end{matrix}\right\}\right]$$

--- --- --- --- --- ---

XB or YB as operand in an adder operation indicates that the element assigned to the X-bus (resp. Y-bus) in a parallel operation of the instruction is to be used also as A-side input. The bus assignment can be explicit or implicit. If no parallel bus assignment is made, a warning message is issued.
Example. XB+U->U, T*MB. T is the A-side adder input, via the X-bus (because of the implicit use of the X-bus in T*MB). Note that if the instruction had been written T+U->U, T*MB, U might have been assigned to the A-side, and T to the B-side.

$$\left\{\begin{matrix} \mathrm{GA} \\ \mathrm{GB} \end{matrix}\right\} + \mathrm{HCO{:}G} \left[\rightarrow \left\{\begin{matrix} \mathrm{U} \\ \mathrm{V} \end{matrix}\right\} \right]$$

$$\left\{\begin{matrix} \mathrm{GA} + \mathrm{GB} \\ \mathrm{GB} + \mathrm{GA} \end{matrix}\right\} + \mathrm{HCO{:}\ G} \left[\rightarrow \left\{\begin{matrix} \mathrm{U} \\ \mathrm{V} \end{matrix}\right\} \right]$$

$$\left\{\begin{matrix} \mathrm{GA} - \mathrm{GB} \\ \mathrm{GB} - \mathrm{GA} \end{matrix}\right\} -1 + \mathrm{HCO{:}G} \left[\rightarrow \left\{\begin{matrix} \mathrm{U} \\ \mathrm{V} \end{matrix}\right\} \right]$$

2.3. H-ADDER OPERATIONS

$$\left\{\begin{matrix} \mathrm{HA\ OP\ HB} \\ \mathrm{HB\ OP\ HA} \end{matrix}\right\} \mathrm{:H} \left[\rightarrow \left\{\begin{matrix} \mathrm{V} \\ \mathrm{W} \end{matrix}\right\} \right]$$

where HA = {MPL, MPR, value, X, XC, DX, IO, XB, Y, YC, DY, DA, T, U, V, W, DCMV, SCLV, RL, RR, YB, FILE}

HB = {0, U, V, W}

and OP = {'+', '-', 'OR', 'AND', 'XOR'}

Special Provisions:

$$\left\{\begin{matrix} \mathrm{HA} + \mathrm{HB} \\ \mathrm{HB} + \mathrm{HA} \end{matrix}\right\} + \mathrm{LGCO} \quad \mathrm{:H} \left[\rightarrow \left\{\begin{matrix} \mathrm{V} \\ \mathrm{W} \end{matrix}\right\} \right]$$

$$\left\{\begin{matrix} \mathrm{HA} - \mathrm{HB} \\ \mathrm{HB} - \mathrm{HA} \end{matrix}\right\} -1 + \mathrm{LGCO} \quad \mathrm{:H} \left[\rightarrow \left\{\begin{matrix} \mathrm{V} \\ \mathrm{W} \end{matrix}\right\} \right]$$

$$\left\{\begin{matrix} \mathrm{HA} \\ 0 \\ \mathrm{HB} \end{matrix}\right\} \mathrm{:H} \left[\rightarrow \left\{\begin{matrix} \mathrm{V} \\ \mathrm{W} \end{matrix}\right\} \right]$$

$$\left\{\begin{matrix} \mathrm{HA} \\ \mathrm{HB} \end{matrix}\right\} + \mathrm{LGCO} \quad \mathrm{:H} \left[\rightarrow \left\{\begin{matrix} \mathrm{V} \\ \mathrm{W} \end{matrix}\right\} \right]$$

2.4. FG-ADDER OPERATIONS

$$\left\{\begin{array}{l} FA//GA \pm T//GB \\ T//GB \pm FA//GA \end{array}\right\} :FG \;\rightarrow\; \left\{\begin{array}{l} TU \\ TV \\ UV \end{array}\right\}$$

where // is the concatenation operator,
FA chosen as in 2.1, and GA, GB chosen as in 2.2.

Example: MPLMPR + TU: FG -> TU is equivalent to:

MPL + T:F -> T,
MPR + U:G -> U,
FCI = GCO

Special Provisions:

$$T//GB{:}FG \;\rightarrow\; \left\{\begin{array}{l} TU \\ TV \\ UV \end{array}\right\}$$

2.5. GH-ADDER OPERATIONS

$$\left\{\begin{array}{l} GA//HA \pm GB//HB \\ GB//HB \pm GA//HA \end{array}\right\} :GH \quad \left[\rightarrow \left\{\begin{array}{l} UV \\ UW \\ VW \end{array}\right\}\right]$$

GA,GB chosen as in section 2,2; HA,HB chosen as in section 2.3

Special provisions:

$$\left\{\begin{array}{l} GA//HA + GB//HB \\ GB//HB + GA//HA \end{array}\right\} + LGCO \quad :GH \quad \left[\rightarrow \left\{\begin{array}{l} UV \\ UW \\ VW \end{array}\right\}\right]$$

$$\left\{\begin{array}{l} GA//HA - GB//HB \\ GB//HB - GA//HA \end{array}\right\} -1 + LGC0 \quad :GH \quad \left[\rightarrow \left\{\begin{array}{l} UV \\ UW \\ VW \end{array}\right\}\right]$$

2.6. STANDARD ADDER OPERATIONS

Adder Operation -> T (F is used)

$$\text{Adder Operation} \rightarrow \left\{\begin{matrix} U \\ V \end{matrix}\right\} \quad \text{(G is used)}$$

Adder Operation -> W (H is used)

$$\text{Adder Operation} \rightarrow \left\{\begin{matrix} TU \\ TV \end{matrix}\right\} \quad \text{(FG is used)}$$

$$\text{Adder Operation} \rightarrow \left\{\begin{matrix} UV \\ UW \\ VW \end{matrix}\right\} \quad \text{(GH is used)}$$

2.7. STANDARD ADDER-IMPLEMENTED MOVES

0 -> T (F is used)

$$\left\{\begin{matrix} A \\ B \end{matrix}\right\} \rightarrow \left\{\begin{matrix} U \\ V \end{matrix}\right\} \quad \text{(G is used)}$$

$$\left\{\begin{matrix} A \\ B \end{matrix}\right\} \rightarrow W$$

(H is used unless A = {Y,YC,DY,DA, RL,RR,SCLV,value}. Then, no adder is used)

$$\left\{\begin{matrix} GA//HA \\ GB//HB \end{matrix}\right\} \rightarrow \left\{\begin{matrix} UW \\ UV \\ VW \end{matrix}\right\} \quad \text{(GH is used)}$$

3. REGISTER OPERATIONS

3.1. T - OPERATIONS

A -> T WHERE A = {X, XC, DX, IO, U, V, value, DCMV, 0}

3.2. U - OPERATIONS

A -> U WHERE A = { 0, T, V, W, MPL, MPR, X, XC, DX, IO, Y, YC, DY, DA, DCMV, SCLV, RL, RR, value}

3.3. V - OPERATIONS

A -> V WHERE A = { 0, T, U, W, MPL, MPR, X, XC, DX, IO, Y, YC, DY, DA, DCMV, SCLV, RL, RR, value}

3.4. W - OPERATIONS

A -> W WHERE A = {Y, DY, YC, DA, RL, RR, U, V, SCLV, value, 0, MPL, MPR, X, XC, DX, IO, DCMV, T}

Notes: In sections 3.1 - 3.4 the following precedences apply:

1). The operation will be performed using a bus (but no adder) if possible. (If it is desired to have the operation performed using an adder, the appropriate adder should be shown explicitly - e.g., U:H->W).

2) If the operation cannot be performed using a bus only, or if the relevent bus already has a conflicting assignment for the instruction, the operation will be performed in an adder, possibly with the use of a bus.

3) The H-adder output cannot go into both V and W in the same instruction. The F-adder output cannot go into both T and U in one instruction.

3.5. XL - OPERATIONS

A -> XL WHERE A = {X, XC, DX, IO, value, T, U, V, DCMV}

3.6. XC - OPERATIONS

A -> XC WHERE A = {X, XA, DX, IO, value, T, U, V, DCMV}

3.7. YL - OPERATIONS

```
A -> YL   WHERE  A = {Y, YC, DY, DA, value,
                     U, V, W, SCLV, RL, RR}
```

3.8. YC - OPERATIONS

```
A -> YC   WHERE  A = {Y, YA, DY, DA, value,
                     U, V, W, SCLV, RL, RR}
```

4. MEMORY-ADDRESS OPERATIONS

4.1. XA - OPERATIONS

```
INC XA
INC XA(A)
DEC XA
DEC XA(A)
B -> XA
CLR XA
```

```
WHERE  A = {2, XC, XC+1, XC-1, Y, YC, DY, DA, U, V, W,
            SCLV, RL, RR, value}

AND    B = {XC, Y, YC, DY, DA, value, U, V, W, SCLV, RL,
            RR}
```

4.2. YA - OPERATIONS

```
INC YA
INC YA(A)
DEC YA
DEC YA(A)
B -> YA
CLR YA
```

```
WHERE   A = {2, YC, YC+1, YC-1, X, XC, DX, IO, value,
             T, U, V, DCMV}

AND     B = {YC, X, XC, DX, IO, value, T, U, V, DCMV}
```

5. X-BUS

```
XB = A   WHERE  A = {X, XC, DX, IO, value,
                     T, U, V, DCMV}
```

6. Y-BUS

```
YB = A   WHERE  A = {Y, YC, DY, DA, value,
                    U, V, W, SCLV, RL, RR}
```

7. R - OPERATIONS

```
    INC RA
    A -> RA
    A -> RC

WHERE   A = {Y, YC, DY, DA, value, U, V, W, SCLV,RL, RR}
```

8. ARRAY MEMORY OPERATIONS

```
A -> X   WHERE  A = {XC, DX, IO, value, T, U, V, DCMV}

A -> Y   WHERE  A = {YC, DY, DA, value,
                     U, V, W, SCLV, RL, RR}
```

9. PROGRAM CONTROL

```
GOTO psa

CALL psa

RTN

EXEC MACRO

IF J > 0 PSB = val

IF STAT = 0  GOTO psa

IF STAT = 1  GOTO psa

IF J > 0  GOTO psa  (NOTE: THIS OPERATION MUST BE
          COMBINED IN AN INSTRUCTION WITH 'DEC J')
```

IF $\left\{\begin{matrix} G \\ H \\ GH \end{matrix}\right\}\left\{\begin{matrix} = \\ < \\ > \end{matrix}\right\}$ 0 GOTO psa

IF SHIFT $\left\{\begin{matrix} \text{SMALL} \\ \text{OVFL} \end{matrix}\right\}$ GOTO psa

Note: In the above operations, ´psa´ is an expression (involving instruction labels, etc.) which reduces to a relocatable program source address. See Figure 4 for a detailed description of these operations.

10. IO OPERATIONS

INT HOST

CLR SBIT

CLR S

A -> IO WHERE A = {X, XC, DX, value, T, U, V, DCMV}

XB = IO (See 5, above)

IOC

INT ENABLE

INT DISABLE

11. COUNTER AND DEVICE OPERATIONS

DEC J

A -> J

A -> DEV

WHERE A = {Y, YC, DY, DA, value, U, V, W, SCLV, RL, RR}

12. D MEM AND DA OPERATIONS

READ

READ (VAL)

READ (VAL:D)

WRITE A

WRITE AB

WRITE (VAL) A

```
WRITE  (VAL:D) AB

INC DA

INC DA(B)

B -> DA

B -> DA (VAL)

B -> DA (VAL:D)

YB = DA                       (See 6, above)

YB = DA(VAL)

YB = DA(VAL:D)
```

In the above expressions, VAL, A and B must satisfy:

a) 0 <= VAL <= ´15´D

b) A = {X, XC, DX, value, T, U, V, DCMV}

c) B = {Y, YC, DY, value, U, V, W, SCLV, RL, RR}

Notes on DMEM and DA Operations:

1. VAL selects one of 16 address registers in DA, the D-memory controller. The register contains the address in D memory to or from which data is to be transferred. The address in the currently selected register can be set with a ´B -> DA´ operation.

2. Commands which select a DA register -- e.g. READ(VAL), YB=DA(VAL:D) -- also set the ´transfer mode´. VAL:D in such formats specifies double word transfer; when ´:D´ is absent, mode of transfer depends upon value. If 15<VAL<32, mode is double; otherwise mode is single. The transfer mode is maintained for all subsequent READ or WRITE operations until a new DA register and mode are selected.

It is the programmer´s responsibility to insure that READ and WRITE commands which do not set a transfer mode are compatible with the mode which will be in force when the instruction is executed.

3. After a READ or WRITE of D memory, the address is updated. If ´INC DA(B)´ or ´B -> DA´ is an operation of the instruction, the transfer address is updated accordingly. Otherwise, in single word mode the transfer

address is incremented by 1, and in double word mode the transfer address is incremented by 2.

4. INC DA(B) must be combined in an instruction with READ or WRITE (in some form).

GLOSSARY

ABS	-- Absolute value.
CLR	-- Clear (Set a register to 0).
DA	-- Collection of 16 cells which act as address registers for D-memory.
DEC	-- Decrement (Subtract from the contents of a register)
DCMV	-- Decimation of V-register. (Bit reversal)
DEV	-- Device register
DMEM	-- Main random access memory.
DX	-- Register into which data from DMEM is placed after a READ operation. (All single word READS or the high-order 16 bits of a double word read)
DY	-- Register into which data from the low-order 16 bits of a double word READ is placed.
EXEC	-- Execute (a MACRO)
F	-- F - adder
FA	-- A-input to the F-adder.
FCI	-- F-adder carry-in.
FG	-- Linked F and G adders
FR FRAC	-- Multiplication descriptor - Inputs and product in fractional form
G	-- G-adder
GA	-- A-input to G-adder.
GB	-- B-input to G-adder.
GCI	-- G-adder carry-in.
GCO	-- G-adder carry-out.
GH	-- Linked G and H adders.
H	-- H-adder.

HA	-- A-input to H-adder.	
HB	-- B-input to H-adder.	
HCI	-- H-adder carry-in.	
HCO	-- H-adder carry-out.	
INC	-- Increment (Add to contents of a register).	
INT	-- Interrupt.	
INTS	-- Interrupts.	
IO	-- Input/output	
J	-- Tally register.	
LGCO	-- Last G-adder carry-out (carry-out from the G-adder operation of the previous instruction).	
MA	-- A-input to multiplier	
MB	-- B-input to multiplier.	

ML }
MPL }
	-- Multiplier product left
	High-order 16 bits	

MR }
MPR }
	-- Multiplier product right
	Low-order 16 bits	

OVFL }
OV }
	_ Absolute value on YB>32 (Used in SHIFT test)	

P2	-- positive x 2's complement	multiplication descriptor
PP	-- positive x positive	
2P	-- 2's complement x positive	
22	-- 2's complement x 2's complement	
PSA	-- Program source address, A-register.	
PSB	-- Program source address, B-register.	
RMEM	-- Read-only-memory	
RA	-- Register containing RMEM address	
RC	-- Register containing RMEM address increment	
RL	-- RMEM left. High order 16 bits of currently adressed word in RMEM.	

RR	-- RMEM right. Low order 16 bits of currently addressed word in RMEM.
RTN	-- Return.
S	-- Status register.
SCLV	-- Register containing the scale (number of leading zeros, exclusive of sign bit) of the contents of the V-register.
SHIFT	-- multiplier used to shift (normalize, align,scale) a number.
SMALL } SM	-- value on YB in range [-16,15] (used in SHIFT test)
STAT	-- Device condition which can be tested.
T	-- Arithmetic register.
U	-- Arithmetic register.
V	-- Arithmetic register
value	-- *signed 16-bit constant*
val	-- non-negative constant
W	-- Arithmetic register.
X	-- Array memory.
XA	-- X-memory address register
XB	-- X-bus.
XC	-- Auxilliary X-memory address register.
XL	-- X-latch. Auxilliary multiplier input register.
Y	-- Array memory.
YA	-- Y-memory address register
YB	-- Y-bus.
YC	-- Auxilliary Y-memory address register.
YL	-- Y-latch. Auxilliary multiplier input register.

FIGURE 8. MACRO OPERATION STRUCTURE

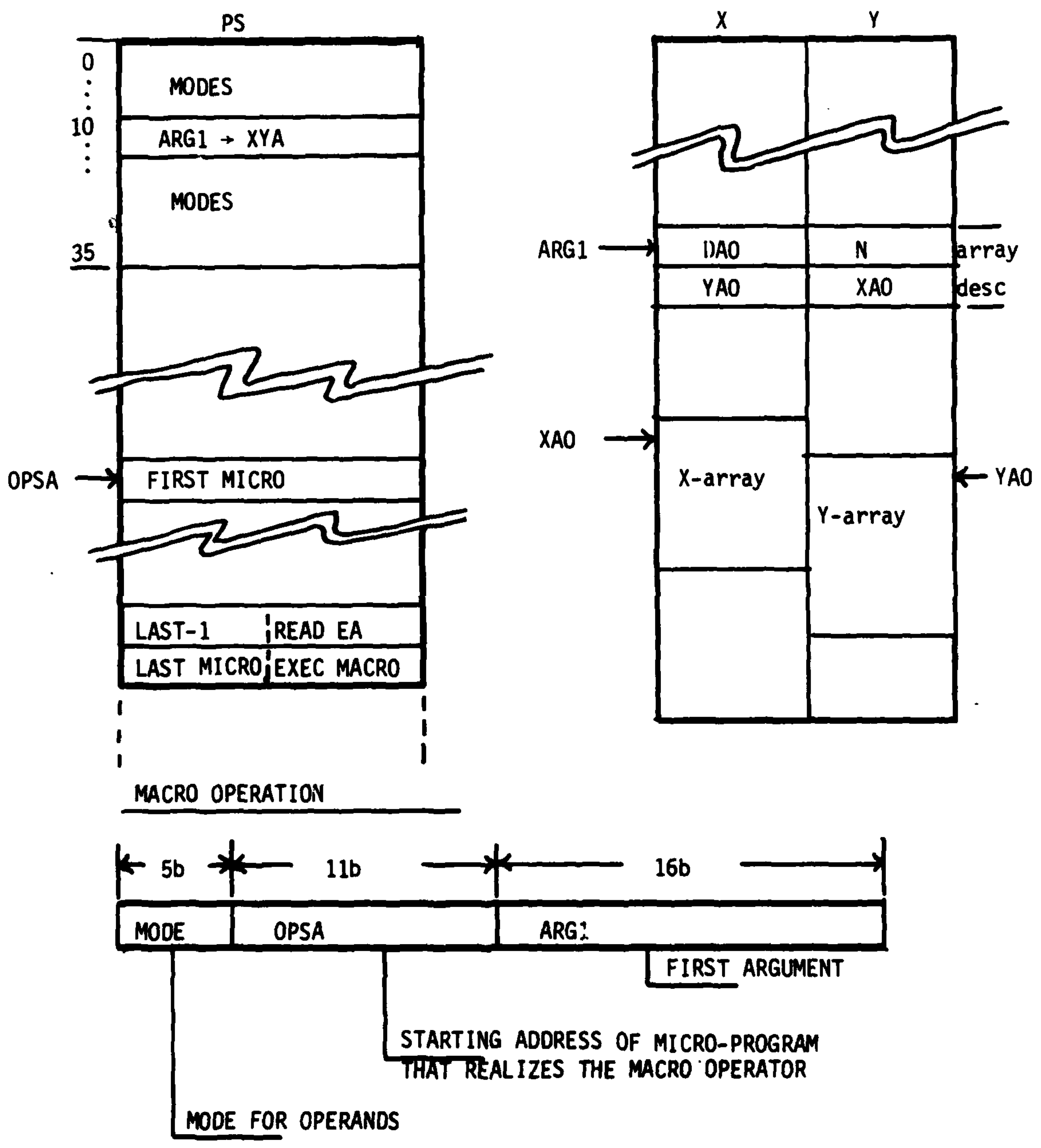


FIGURE 9. <u>CHI-5 MACRO OPERAND MODES</u>

MODE		
0	D(ARG1)→W	"D DIRECT
1	D(ARG1)→W; READ	
2	ARG1 W	"IMMEDIATE
3	ARG1 W; READ	
4	X(ARG1)→W	"X DIRECT
5	X(ARG1)→W; READ	
6	Y(ARG1)→W	"Y DIRECT
7	Y(ARG1)→W; READ	
10	ARG1→XA,YA	"SET XYA
11	ARG1→XA,YA; READ	
12	ARG1→XA	"SET XA
13	ARG1→XA; READ	
14	ARG1→YA	"SET YA
15	ARG1→YA; READ	
16	ARG1→J	"SET J
17	ARG1→J; READ	
20		
21		
22	ARG1→DA(12)	"SET DA(12)
23	ARG1+DBLE→DA(12)	"SET DA(12) DOUBLE
24	ARG1+U→W	"U INDEXED
25	ARG1+U→W; READ	
26	ARG1+V→W	"V INDEXED
27	ARG1+V→W; READ	
30	PREINC(ARG1)	"DA(17)+ARG1→DA(17) READ(17)
31	POSTINC(ARG1)	"READ(17) DA(17)+ARG1→DA(17)
32	INDEX READ	"D(DA(17)+ARG1)→DX,DY
33	DBL INDEX READ	"D(DA(17)+ARG1+U)→DX,DY
34	DBL INDEX	"DA(17)+ARG1+U→DA(17)-U-ARG1→W
35	NOOP	
36	PS→D	"READ PS
37	D→PS	"WRITE PS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MDA 903-82-C-0136-01	2. GOVT ACCESSION NO. AD-A114 707	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CHI-5 Micro-Programming Reference Manual		5. TYPE OF REPORT & PERIOD COVERED Quarterly Technical Report 10 Feb 1982 - 10 May 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. B. Bruckner		8. CONTRACT OR GRANT NUMBER(s) MDA 903-82-C-0136
9. PERFORMING ORGANIZATION NAME AND ADDRESS CHI Systems, Inc. 100 Edward Burns Place Goleta, CA 93117		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA ORDER NO. 3625
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Supply Service - Washington Room 1D245 , The Pentagon Washington , D.C. 20110 (T. Busnell)		12. REPORT DATE 10 May 1982
		13. NUMBER OF PAGES 54
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DCASMA Van Nuys (S0512A) 6230 Van Nuys Blvd. Van Nuys, CA 91408		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Assembler, Array Processor, Microcode

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The CHI-5 is a general purpose computer whose inner structure is that of an array processor. This document is a micro-programming reference manual for the CHI-5. It includes descriptions of the major logical modules of the CHI-5, a description of the CHI-5 micro-assembler, and specifications of the assembler syntax for the CHI-5 micro-operations

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)